*PROJECT REPORT*

**PROGRAM LOGIC**

This project uses Clojure via InelliJ IDE with the Cursive add-on. This program will sort a large list of integers from the provided .txt file which contains one million integer values. The program uses the merge sort algorithm and sorts from smallest to largest. The program will then run the same algorithm with 2, 4, 8, 16, 32, and 64 threads.

It will measure the time needed to sort the .txt file using parallelism and the above number of threads. The purpose of this report is to explain the findings of this code.

***CODE DESCRIPTION***

*mrg(x, y)*

This function serves to merge two different merge 2 different parts of an array. The parameters x and y represent two different arrays that can be merged. This function is recursive and will continue merging the lists as required. It will concatenate the files once completed.

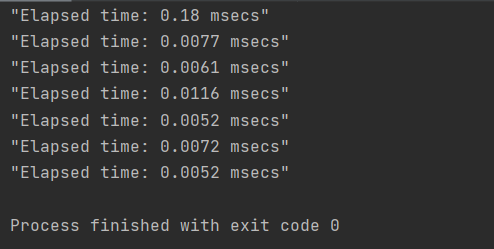
*mergeSort(input)*

This function serves to begin sorting process but to only one thread. It makes use of map to transform data into new data by applying the mergeSort to same function to each element.

*mergeSort-2thread (input) – mergeSort-64thread (input)*

These functions serve to increase the number of threads when using mergeSort. They all use the same type of approach and is written nearly the same as mergeSort, except it makes use of pmap instead of map. Pmap is nearly the same as using a map function, however, f is applied in parallel instead.

*Sample Output*

**

***TESTING RESULTS***

This code has been run 3 times and the average of each result has been plotted in the graph below

To help make sense of the data, I will explain the data for each file.

|  |  |
| --- | --- |
| Average run time for 100lines.txt file | |
| Number of Threads | Average Run Time |
| 1 | .222 |
| 2 | .0072 |
| 4 | .0214 |
| 8 | .0016 |
| 16 | .0031 |
| 32 | .0052 |
| 64 | .0054 |

This file contained 100 numbers from the numbers.txt file. This was done to ensure that my code would run as intended and to see if my results would vary depending on how large the file was. Using this file size, you stop seeing improvements around 8 threads. The time begins increasing as you add more threads.

|  |  |
| --- | --- |
| Average run time for 3000lines.txt file | |
| Number of Threads | Average Run Time |
| 1 | .1841 |
| 2 | .0070 |
| 4 | .0056 |
| 8 | .0021 |
| 16 | .0072 |
| 32 | .0233 |
| 64 | .0172 |

This file contained 3000 numbers from the numbers.txt file. This was done to see how much of a difference adding a few thousand numbers would make to this code. Using this file size, you stop seeing improvements around 8 threads. The time begins increasing as you add more threads.

|  |  |
| --- | --- |
| Average run time for numbers.txt file | |
| Number of Threads | Average Run Time |
| 1 | .1841 |
| 2 | .0059 |
| 4 | .0055 |
| 8 | .0051 |
| 16 | .0049 |
| 32 | .0049 |
| 64 | .0142 |

This file contained 100 numbers from the numbers.txt file. This was done to ensure that my code would run as intended and to see if my results would vary depending on how large the file was. Using this file size, you stop seeing improvements around 16 to 32 threads. The time begins increasing as you add more threads.

***CONCLUSIONS***

Interestingly enough, the 100 line file took the longest amount of time to run the first thread and both this file and the 3000 line file stopped seeing improvement when the threads increase at eight threads. This result may be because of the size of file; it is simply not large enough to require so many threads. Imagine that this file represents a hallway, and the number of threads represents the number of sweepers we have. We want to sweep this from one end to the other. One cleaner will take a while to clean the hallway alone. If we add a cleaner to work side by side to the first cleaner, it will get done a bit faster, however, the hallway is only so wide. Adding too many cleaners will begin to cause a hinderance and not really improve any cleaning.

The numbers file stopped showing improvements a bit later at 32 threads. This is because the file contained much more data and would make better use of more threads. Returning to the hallway analogy, we can image that the hallway’s width was increased. Due to the increase, more workers can stand side by side before they begin hindering each other.

***REFERENCES***

*Resources Used*

Stuart Sierra, S. H. (n.d.). *API for clojure.string - Clojure v1.10.2 (stable)*. Retrieved from Clojure: https://clojure.github.io/clojure/clojure.string-api.html

https://clojuredocs.org/clojure.core

https://clojuredocs.org/clojure.core/pmap

Retrieved from time https://clojuredocs.org/clojure.core/time

https://stackoverflow.com/questions/55770730/trying-to-read-a-text-file-in-clojure-and-insert-the-data-into-a-list-or-a-vecto

*Code References*

https://gist.github.com/baabelfish/6573984